An Experiment in Determining Software Reliability Model Applicability

Allen P. Nikora
Jet Propulsion Laboratory/
California Institute of Technology

Michael R. Lyu
Information Sciences and Technologies
Research Lab
Bell Communications Research

1. Introduction

Most of the reports on experience with software reliability models have made use of data from actual software development efforts [4, 7, 10]. These reports have been useful in helping practitioners determine the behavior the models will exhibit under actual test and operational conditions. However, the failure data from actual projects is influenced by process and product characteristics that are not taken into account in most models. The shape of a failure intensity curve may be influenced by the complexity of the software under test, the number of testers available, the skill level of the testing staff, and the type of testing being performed (e.g. functional testing, path testing, data coverage testing). These data are also subject to uncertainty and distortion. The complexity of real world failure data may obscure properties of software reliability models that might be revealed by executing the models on simpler data sets, We have created sets of interfailure times by generating sequences according to the distributions for two of the more-widely used software reliability models, and have executed six models on each data set. The results of this activity have suggested ways in which existing methods might be most effectively used in choosing the most appropriate model. In the remaining sections we discuss the following items:

1. The way in which the data sets were created, and the way in which the models were executed using these data sets.
2. The results of the experiment.
3. A discussion of the results, and recommendations for further work.

11. Method

In creating the data sets, we generated 40 sequences of interfailure times according to the method discussed in Section 12.3 of [9]. The first twenty sequences were created by drawing random samples from an exponential distribution, while the remaining twenty were created to simulate sequences of interfailure times for a logarithmic Nonhomogeneous Poisson Process. To generate data simulating a logarithmic Nonhomogeneous Poisson Process, the following was done for each sequence:

1. Generate a sequence of random numbers, z_i, uniformly distributed in the interval (O, 1).
2. For each z_i, compute an exponential random value $u_i = -\ln (1 - Z_i)$. This represents the i'th failure interval for a stationary Poisson process with a rate of 1.
3. Set vi= $\sum_i^i u_i$. This represents the i'th failure time for the Poisson process.
4. Convert the failure time for the stationary Poisson process to the failure time for the logarithmic Poisson process, W_i:

$$W_i = \mu^{-1} \text{ (vi)}$$

where μ is the mean value function for the logarithmic Poisson model:

$$\mu(\tau) = (1/\Theta)\ln(\lambda_0\Theta\tau + 1)$$

λ_0 = initial failure intensity

Θ = failure intensity decay parameter

τ = total elapsed execution time

5. Convert the logarithmic Poisson failure times to failure intervals by taking the differences between successive values of w_i.

A similar process was used to generate the sequences simulating data from an exponential Nonhomogeneous Poisson Process.

Six of the better-known software reliability models were then run on the sequences using maximum likelihood parameter estimation. The six models and their hazard rates or mean value functions are given below. Detailed descriptions may be found in [9] and [11].

1. **Geometric Model** - hazard rate z(t) for time "t" between the i-1 'th and the i'th failure is $z_0\Theta^{i-1}$, where z_0 is the initial hazard rate, and Θ is the decay constant.
2. **Jelinski-Moranda Model** - hazard rate is :

$$z(t) = K\left(\frac{E_T}{I_T} - \varepsilon_C t\right)$$

where $z(\tau)$ represents the hazard rate at failure time τ, K is the proportionality constant, E_T is the number of errors initially in the program, IT is the number of machine instructions in the program, and ε_C is the cumulative number of failures removed in the interval $[0, \tau]$.

3. **Littlewood-Verrall Model (quadratic form)** - the hazard rate for the quadratic form is:

$$z(t_i) = t_i + \beta_0 \frac{a}{\beta_1 i^2}$$

where α, β_0, and β_1 are parameters of the model, i represents the failure number, and t_i is the time between the i-1 'th and i'th failures.

4. **Muss Basic Model** - the mean value function for this model is:

$$\mu(t) = \nu_0\left(1 - \exp\left(-\frac{\lambda_0}{\nu_0} t\right)\right)$$

where $\mu(t)$ represents the mean number of failures at time t, λ_0 represents the initial failure intensity, and ν_0 represents the estimated total number of failures that would be observed over an unlimited amount of execution time.

5. **Muss-Okumoto Model** - the mean value function for this model was given at the start of this section.
6. **Nonhomogeneous Poisson Process (NHPP) Model** - the mean value function for this model is given by:

$$p(t) = a(1 - \exp(-bt))$$

where $\mu(t)$ represents the mean number of failures at time t, a is the estimated total number of failures that would be observed over an unlimited amount of execution time, and b is the intensity decay parameter.

The tool CASRE [6] was used to run the models. CASRE is a software reliability modeling tool implemented in a Microsoft Windows environment. The core modeling capabilities of this tool are the libraries originally implemented for version 5 of the software reliability modeling tool SMERFS [1 2]. The object code for these libraries was linked into the executable CASRE module. The command interface is a set of pull-down menus that make it easy to navigate through the functional areas of the tool. Input data and model results are displayed both as text and as high-resolution graphics that were designed to be easy for non-specialists to interpret. Model results can be written to a text file which can be brought into a spreadsheet, database, or statistical analysis package for further analysis.

An array of estimated Mean Times To Failure (MTTF) was then generated, using the parameter estimates obtained after processing the last observation in each sequence. The Kolmogorov-Smirnov goodness of fit test was used to determine the goodness of fit of the model results to the input data set. To evaluate model applicability, the prequential likelihood, model bias, and bias trend were computed [2]. A brief description of these three criteria is given below:,

1. **Prequential Likelihood** - although similar in form to the likelihood function used for maximum likelihood estimation, this function is not used to estimate model parameters. Rather, the parameter estimates and actual observed failure times are used in this function to compute a value that can be used to determine how much more likely it is that one model will produce accurate estimates than another model. This likelihood is given by the value of the ratio of the prequential likelihoods for the two models being compared.
2. **Model Bias** - this measure uses the estimated probability of failure for each failure interval to determine the extent to which a model introduces bias into its estimates. If a

model is biased, it can be optimistically biased (estimates of MTTFs are higher than what is actually observed), or it can be pessimistically biased. The cumulative distribution function (calf) for the estimated failure probabilities is compared to the cdf for iid random variables in the interval (0,1) using the Kolmogorov-Smirnov (KS) test. The KS test statistic reveals the extent of model bias.

3. **Model Bias Trend** - this measure uses the estimated probability of failure for each failure interval to determine whether model bias changes over time. A model may be optimistically biased during the early stages of testing, while it may become pessimistically biased during the later stages. The analysis is similar to that for model bias, except that the estimated failure probabilities are transformed in a way that preserves temporal information.

The models were then ranked with respect to all four criteria. Model noise was not included in the criteria because, unlike the other criteria, it provides no absolute indicator of how well a model performs, nor does it necessarily measure how well one model performs compared with other models.

111. Results

The results of this experiment are summarized in Tables 1-12 below. The first two tables summarize the performance of all models across the input sequences, Table 1 summarizes the performance of all models across the 20 sequences drawn from an exponential distribution, while Table 2 summarizes the performance of all models across the 20 sequences drawn from the logarithmic Poisson distribution. Tables 1 and 2 are interpreted as follows.

1. Even-numbered columns rank the model across all inputs of a specific type with respect to prequential likelihood, model bias, model trend, and the Kolmogorov-Smirnov goodness-of-fit test. Column 1 gives the mean overall rank of the model, which is computed by equally weighting the individual ranks according to prequential likelihood, model bias, model bias trend, and goodness of fit (KS test).

Model Name	Overall Rank	Std Dev	Preqnt'l Likelihood Rank	Std Dev	Bias Rank	Std Dev	Bias Trend Rank	Std Dev	KS Rank	Std Dev
Geometric	1.950	0.88704	1.550	0.60481	1.750	0.44426	2.4S0	0.68633	2.250	0.78640
Jelinski-Moranda	5.450	0.99868	5.500	0.88852	5.500	0.88852	5.300	1.38031	5.250	1.33278
1.ittlewood-Verrall	2.700	1.08094	3.100	0.71818	3.150	0.67082	2.100	1.44732	1.850	1.34849
Musa Basic	5.900	0.44721	5.950	0.22361	5.950	0.22361	5.900	0.44721	5.950	0.22361
Muss Okumoto	1.050	0.22361	1.500	0.51299	1.250	0.44426	1.800	0.61559	2.200	0.69585
NHPP	5.900	0.44721	5.900	0.44721	5.900	0.44721	5.950	0.22361	6.000	0.00000

Table 2- Summary of model rankings - data representative of logarithmic NHPP model

This information is shown in Figures 1 and 2 below. These figures also show the mean ranks according to prequential likelihood, model bias, model bias trend, and goodness of fit.

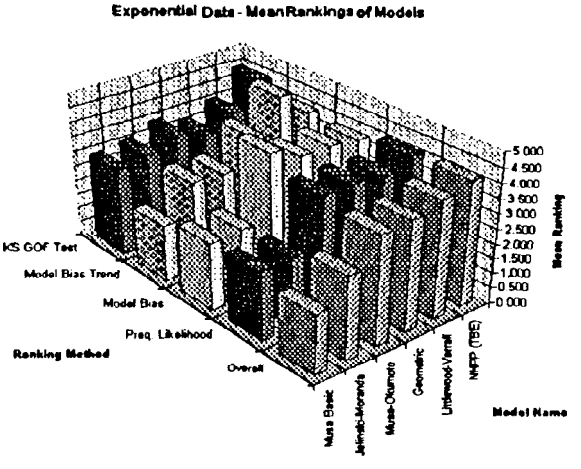

Figure 1 - Mean Model Rankings - Exponential NHPP Inputs

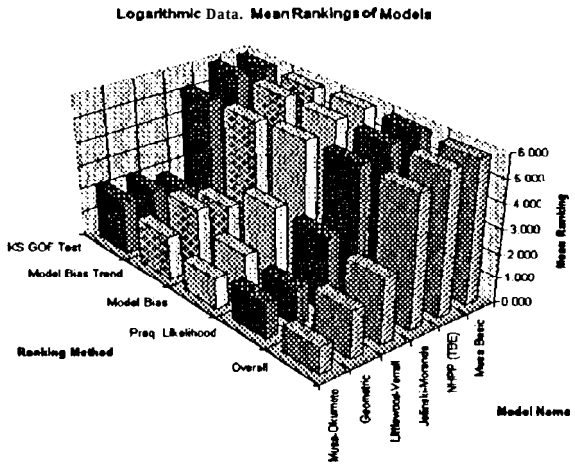

Figure 2- Mean Model Rankings - Logarithmic NHPP Inputs

Tables 3 and 4 summarize the ranking frequencies for each model for a particular input type. For each model that was run using the 20 sets of data drawn from an exponential distribution, Table 3 gives the number of times that model was assigned a particular rank. Table 4 is interpreted the same way, except that the input data to the models was the 20 sequences intended to be representative of a logarithmic Nonhomogeneous Poisson Process.

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Fifth	Times Ranked Sixth
Geometric	2	1	4	8	4	1
Jelinski-Moranda	7	4	4	1	0	4
Littlewood-Verrall	2	1	6	3	3	5
Muss Basic	8	6	4	2	0	0
Muss Okumoto		2	5	7	5	0
NHPP	4	2	2	0	2	10

Table 3- Overall ranking frequency - exponential NHPP inputs

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Fifth	Times Ranked Sixth
Geometric	8	5	7	0	0	0
Jelinski-Moranda	0	0	1	4	0	15
Littlewood-Verrall	2	6	10	1	0	1
Muss Basic	0	0	0	1	0	19
Muss Okumoto	19	1	0	0	0	0
NHPP	0	0	0	1	0	19

Table 4- Overall ranking frequency - data representative of logarithmic NHPP model

We found that traditional goodness-of-fit tests (in this case, the Kolmogorov-Smirnov test) do not seem to be the best way of identifying the most appropriate model. For instance, we would expect that the Jelinski-Moranda, the Muss Basic, of the NHPP model would perform best on the data sequences generated by drawing random samples from an exponential distribution. However, Table 5 below indicates that collectively, these models do not rank first according to this measure as often as the' Geometric, the Littlewood-Verrall, and the Musa-Okumoto.

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Fifth	Times Ranked sixth
Geometric	6	2	3	4	2	3
Jelinski-Moranda	4	4	5	2	3	2
Littlewood-Verrall	3	3	4	3	3	4
Muss Basic	4	5	3	5	2	1
Muss Okumoto	2	3	5	5	4	1
NHPP	2	2	1	4	2	9

Table 5- KS Test ranking frequency - exponential NHPP inputs

We see the same phenomenon in the data sets created to simulate a logarithmic Nonhomogeneous Poisson process. In Table 6 below, note that the Littlewood-Verrall model ranks first 12 times, while the Muss-Okumoto model, which we would expect to perform the best, ranks first only three times. Figures 3 and 4 show this information in the form of a 3-D plot.

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Fifth	Times Ranked sixth
Geometric	4	7	9	0	0	0
Jelinski-Moranda	1	0	0	5	0	14
Littlewood-Verrall	12	3	3	1	0	1
Muss Basic	0	0	0	0	1	19
Muaa Okumoto	3	10	7	0	0	0
NHPP	0	0	0	0	0	20

Table 6- KS Test ranking frequency - data representative of logarithmic NHPP model

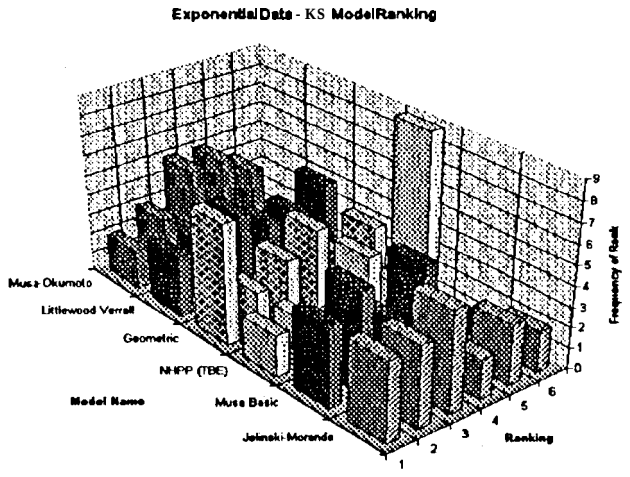

Figure 3- KS Test Model Ranking - Exponential NHPP Inputs

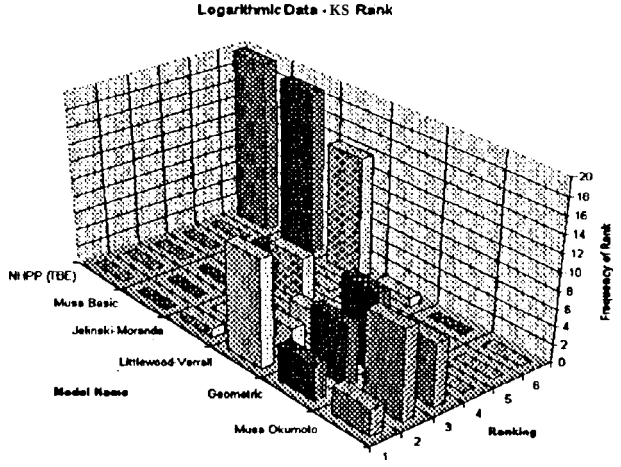

Figure 4- KS Test Model Ranking - Logarithmic NHPP Inputs

Looking at the other methods of ranking the models, we find that they provide results much closer to what we might expect. Specifically, the Jelinski-Moranda, Muss Basic, and NHPP models are favored when using the sequences drawn from an exponential distribution as input, while the Muss-Okumoto model is favored when using data simulating a logarithmic Nonhomogeneous Poisson Process as input. This is shown in tables 7-12, below.

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Fifth	Times Ranked sixth
Geometric	1	3	0	8	8	0
Jelinski-Moranda	11	1	4	1	0	3
Littlewood-Verrall	4	1	6	0	1	8
Muss Basic	4	11	i	3	1	0
Musa Okumoto	o	1	5	8	6	0
NHPP	3	4	0	3	0	10

Table 7- Prequential Likelihood ranking frequency - exponential NHPP inputs

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Fifth	Times Ranked sixth
Geometric	2	1	1	7	9	0
Jelinski-Moranda	10	2	4	2	0	2
Littlewood-Verrall	1	1	8	3	1	6
Muss Basic	3	11	4	2	0	0
Muss Okumoto	1	1	1	6	7	4
NHPP	5	3	1	1	1	9

Table 8- Model Bias ranking frequency - exponential NHPP inputs

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Fifth	Times Ranked Sixth
Geometric	1	3	1	8	7	0
Jelinski-Moranda	2	7	6	1	2	2
Littiewood-Verrall	2	0	1	1	7	9
Muss Basic	12	3	2	2	0	1
Muss Okumoto	2	3	9	6	0	0
NHPP	4	2	2	2	1	9

Table 9- Model Bias Trend ranking frequency - exponential NHPP inputs

Model Name	Times Ranked First	Times Ranked Second	Times Ranked Third	Times Ranked Fourth	Times Ranked Flab	Times Ranked Sixth
Geometric	2		11	0	0	0
Jelinski-Moranda	1	0	1	3	0	15
Littlewood-Verrall	11	1	5	2	0	1
Muss Basic	0	0	0	1	0	19
Muss Okumoto	6	12	2	0	0	0
NHPP	0	0	0	0	1	19

Table 12- Model Bias Trend ranking frequency - simulated logarithmic NHPP data

IV. Discussion

Even though the data sets used as inputs to the models were created to favor either the exponential or logarithmic NHPP models, we have seen that in a few cases, other models were favored over the ones that were expected to be chosen. Along with earlier work in this area [2], this demonstrates that selection of the most appropriate software reliability model for a testing effort must continue after testing has started and model application has begun. Indiscriminate application of statistical methods can result in not choosing the most appropriate model on which to base reliability forecasts. Although a one-step ranking of the models with respect to prequential likelihood, bias, bias trend, and goodness of fit can provide a good idea of the most appropriate model in many cases, there may still be times for which a less appropriate model might be chosen. Part of the problem seems to be that the goodness-of-fit test is not sensitive enough to make find distinctions among models. It is perhaps better suited to serve as a preliminary screening, rejecting models that do not fit to a pre-specified significance level. For those models that do fit the data to the specified significance level, successive application of the other three methods listed above can provide a better idea of which model is more appropriate to the data being analyzed. We recommend following the steps below to choose the most appropriate model.

1. Apply a goodness of fit test to determine if the model results fit the input data to a specified significance level.
2. If more than one set of model results are a good fit to the data:
   a. Choose the most appropriate model(s) based on the prequential likelihood.
   b. In the event of a tie, use the model bias, then model bias trend to break the tie.
   c. Use techniques, such as forming linear combinations of model results [5, 6, 7] or model recalibration [3, 5], to increase the accuracy of the model predictions. Reports on the use of these methods indicate that they can be used to significantly increase the models' predictive accuracy.
3. If only one model provides a good fit to the data, choose that model.
4. If no models provide a good fit to the data:
   a. Choose the most appropriate model(s) based on the prequential likelihood.
   b. Use the linear combination and model recalibration techniques mentioned above to increase the accuracy of model predictions.
   c. Apply the goodness of fit test to the adjusted model results to identify those that are a good fit to the data.

V. Conclusion

We have seen that selection of the most appropriate software reliability model is a process that continues throughout the testing phase. Even if the characteristics of the testing process are well-known for a particular development effort, this is no guarantee that the model whose assumptions appear to best match these characteristics will be the most appropriate model. Finally, a staged application of the model applicability criteria previously discussed appears to be the best way of selecting the most appropriate model. Reliance on a single measure to choose the most appropriate model can lead to making an incorrect choice. Using a weighted ranking scheme involving several criteria can reduce the chances of making an incorrect selection, but the risk of choosing an inappropriate model can still be greater than using the multiple opportunities of a staged ranking process to eliminate less appropriate choices.

Because of time limitations, we were unable to investigate model recalibration and linear combination techniques in this experiment, although we are planning to do so in the future. We are also planning on examining the behavior of the models with respect to other distributions of interfailure times that might be encountered. Further work on identifying the most appropriate model for a development effort is needed. Although prequential likelihood and model bias computation have proven to be useful methods, other approaches, such as the Akaike information criterion [1], should be further investigated.

In addition, more work is needed in relating product and development process characteristics to appropriate models. Although it seems to be the case that there is no way to determine with certainty which model is the most appropriate for a particular effort [2], there may be ways of using measures of the development process and the product characteristics to guide the selection of models that are likely to produce valid predictions. Alternatively, these measures may be incorporated into the model directly so as to more accurately describe the fault detection and removal process [8, 10].

Acknowledgement

The work performed in this paper was partly perfomed at the Jet Propulsion Laboratory, California Institute of Technology, under a NASA contract. The enhancement and maintenance of CASRE is being supported by the US Air Force Operational Test and Evaluation Center.

References

1 H. Akaike, "A New Look at Statistical Model Identification," IEEE Transactions on Automatic Control, vol. AC-1 9, pp. 716-723, 1974.

2 A. A. Abdel-Ghaly, P. Y. Chan, B. Littlewood, "Evaluation of Competing Software Reliability Predictions," IEEE Transactions on Software Engineering, vol. SE-12, pp.950-967, September, 1986.

3 S. Brocklehurst, P. Y. Chan, B. Littlewood, J. Snell, "Recalibrating Software Reliability Models," IEEE Transactions on Software Engineering, vol, SE-16, no. 4, pp. 458-470, April, 1990.

4 N. Karunanithi, D. Whitley, Y. Malaiya, "Using Neural Networks in Reliability Prediction", IEEE Software, vol. 9, no. 4, pp 53-59, July, 1992

5 M. Lu, S. Brocklehurst, B. Littlewood, "Combination of Predictions Obtained from Different Software Reliability Growth Models", in Proceedings of the Tenth Annual Software Reliability Symposium, Phipps Mansion, Denver, CO, June, 1992

6 M, R. Lyu, A. Nikora, "Software Reliability Measurements Through Combination Models: Approaches, Results, and a Case Tool," in Proceedings of the 15th Annual International Computer Software and Applications Conference (COMPSAC91), Tokyo, Japan, September, 1991.

7 M. R. Lyu, A. Nikora, "Applying Reliability Models More Effectively," IEEE Software, vol. 9, no. 4, pp 43-52, July, 1992.

8 J. C. Munson, T. M. Khoshgoftaar, "The Use of Software Complexity Metrics in Software Reliability Modeling", in Proceedings of the 1991 International Symposium on Software Reliability Engineering, Austin, TX, May, 1991

9 J. D. Muss, A. Iannino, K. Okumoto, Software Reliability: Measurement. Prediction. Application, McGraw-Hill, New York, 1987

10 A. Nikora, "Early Prediction of Software Reliability from Product and Process Characteristics", Ph.D. research proposal, submitted to the Department of Computer Science, University of Southern California, January, 1994

11 ANSI/AIAA R-O] 3-1992, “Recommended Practice for Software Reliability”, sponsored by American Institute of Aeronautics and Astronautics, February, 1993.

12 Farr, W. H., “Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) User’s Guide”, Revision 3, September, 1993